**操作系统上机报告**

班级： 191174

学号： 20161001764

姓名： 牟鑫一

日期： 2019/06/04

**实验一 多级队列调度算法**

* 1. **实习题目**

用多级队列调度算法处理多个就绪队列。

* 1. **程序功能**

分别利用轮转法和短进程优先调度算法处理对应的进程队列，并输出各进程运行的时间。

* 1. **设计思路**

系统将所有的就绪进程按先来先服务算法的原则，排成一个队列，每次调度时，系统把处理机分配给队列首进程，并让其执行一个时间片。当执行的时间片用完时，由一个计时器发出时钟中断请求，调度程序根据这个请求停止该进程的运行，将它送到就绪队列的末尾，再把处理机分给就绪队列中新的队列首进程，同时让它也执行一个时间片。

给CPU设置时间片，CPU按进程排列顺序依次用一个时间片的时间片的时间来处理每个进程，若一个时间片内该进程未被处理完，则将剩余部分保存在队列尾部；若一个时间片内该进程被处理完，则保存该进程的运行时间=前几个未单次处理完毕的进程数\*时间片+该进程须运行时间，将该进程的须运行时间改为0。进程处理循环过程中若遇须运行时间=0的进程，跳过。

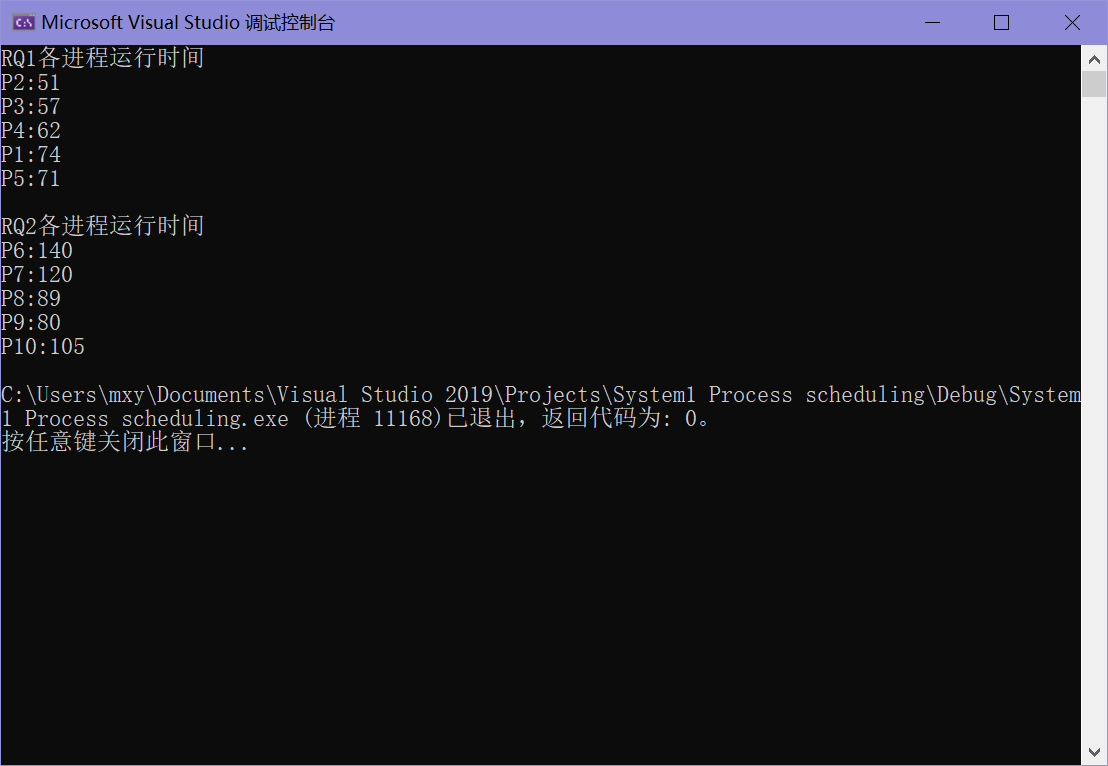
短进程优先调度算法，进程按须运行时间来划分优先级，时间短的优先级高，优先级高的进程先被处理。

* 1. **源代码**

1. #include <iostream>
2. #include <fstream>
3. #include <sstream>
4. #include <string>
5. **using** **namespace** std;
7. **struct** LinkNode {                          //定义链表结点
8. string name;                           //进程名
9. **int** need;                              //需运行时间
10. **int** turn;                              //周转时间
11. LinkNode\* link;
12. LinkNode() {};
13. LinkNode(string na, **int** ne, **int** tu)
14. {
15. name = na;
16. need = ne;
17. turn = tu;
18. link = NULL;
19. }
20. };
22. LinkNode\* RQ1, \* RQ2;
24. **int** main() {
25. **int** i = 0, j = 0;
26. string A[3][5];
27. string B[3][5];
28. LinkNode\* p, \* q, \* s = **new** LinkNode;
29. **int** clock = 0;         //时钟
30. **int** pie = 7;            //时间片
32. ifstream ifile;
34. ifile.open("RQ1.data");//将文档中的数据存入数组A，方便将其插入链表RQ1中
35. string sline;
36. string out;
37. string s1, s2, s3, s4, s5;
38. **while** (getline(ifile, sline))
39. {
40. istringstream sin(sline);
41. sin >> s1 >> s2 >> s3 >> s4 >> s5;
42. A[i][0] = s1;A[i][1] = s2;A[i][2] = s3;A[i][3] = s4;A[i][4] = s5;
43. i++;
44. }
45. ifile.close();
47. ifile.open("RQ2.data");//将文档中的数据存入数组B，方便将其插入链表RQ2中
48. **while** (getline(ifile, sline))
49. {
50. istringstream sin(sline);
51. sin >> s1 >> s2 >> s3 >> s4 >> s5;
52. B[j][0] = s1;B[j][1] = s2;B[j][2] = s3;B[j][3] = s4;B[j][4] = s5;
53. j++;
54. }
55. RQ1 = (LinkNode\*)malloc(**sizeof**(LinkNode));    //建立RQ1链表
56. **if** (RQ1)
57. {
58. string name1 = A[0][0];
59. **int** need1 = atoi(A[1][0].c\_str());
60. **int** turn1 = atoi(A[2][0].c\_str());
61. RQ1 = **new** LinkNode(name1, need1, turn1);
62. p = RQ1;
63. **for** (**int** j = 1; j < 5; j++) {
64. name1 = A[0][j];
65. need1 = atoi(A[1][j].c\_str());
66. turn1 = atoi(A[2][j].c\_str());
67. p->link = **new** LinkNode(name1, need1, turn1);
68. p = p->link;
69. }
70. }
71. RQ2 = (LinkNode\*)malloc(**sizeof**(LinkNode));     //建立RQ2链表
72. **if** (RQ2)
73. {
74. string name2 = B[0][0];
75. **int** need2 = atoi(B[1][0].c\_str());
76. **int** turn2 = atoi(B[2][0].c\_str());
77. RQ2 = **new** LinkNode(name2, need2, turn2);
78. p = RQ2;
79. **for** (**int** j = 1; j < 5; j++) {
80. name2 = B[0][j];
81. need2 = atoi(B[1][j].c\_str());
82. turn2 = atoi(B[2][j].c\_str());
83. p->link = **new** LinkNode(name2, need2, turn2);
84. p = p->link;
85. }
86. }
87. p = RQ1;      //利用轮转法处理RQ1
88. **int** a = 0;
89. **while** (p != NULL) {
90. **while** (p != NULL && p->need > 0) {
91. q = p;
92. **while** (q->link != NULL)q = q->link;
93. **if** (p->need > pie) {
94. clock += pie;
95. p->need -= pie;
96. **if** (s != NULL) s->link = p->link;
97. q->link = p;
98. p = p->link;
99. q->link->link = NULL;
100. }
101. **else** {
102. a++;
103. clock += p->need;
104. p->need = 0;
105. p->turn += clock;
106. **if** (a == 1) RQ1 = p;
107. s = p;
108. p = p->link;
109. }
110. }
111. **while** (p != NULL && p->need == 0) {
112. p = p->link;
113. }
114. }
115. **for** (**int** b = 0; b < 5; b++) {
116. q = RQ2;
117. p = (LinkNode\*)malloc(**sizeof**(LinkNode));
118. **if** (p)
119. {
120. p->need = 1000;
121. p->link = NULL;
122. **while** (q != NULL) {
123. **if** (q->need != 0 && q->need < p->need) p = q;
124. q = q->link;
125. }
126. clock += p->need;
127. p->turn += clock;
128. p->need = 0;
129. }
130. }
131. p = RQ1;
132. cout << "RQ1各进程运行时间" << endl;
133. **while** (p != NULL) {
134. cout << p->name << ":" << p->turn << endl;
135. p = p->link;
136. }
137. q = RQ2;
138. cout << "RQ2各进程运行时间" << endl;
139. **while** (q != NULL) {
140. cout << q->name << ":" << q->turn << endl;
141. q = q->link;
142. }
143. **return** 0;
144. }
     1. **测试**

测试数据如下：RQ1: P1-P5, RQ2: P6-P10

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 进程 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 运行时间 | 16 | 11 | 14 | 13 | 15 | 21 | 18 | 10 | 7 | 14 |
| 已等待时间 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 |



* 1. **实习心得**

为了方便数据的导入，使用了一个二维数组临时存储数据，然后再用二维数组中的数据构建链表RQ1和RQ2，指针的问题是最为麻烦的，有时候先改数据，结果却改了指针，有时候想改指针，却又只是修改了数据，期间通过p、q两个指针链接链表，两个指针同时指向了同一个结点，导致本只想修改一个指针的指向，却一改另一个也跟着改了，在这个问题上花了很多时间，当然现在对指针的理解也更加深刻了。

**实验二 银行家算法**

* 1. **实习题目**

利用银行家算法来避免死锁的发生。

* 1. **程序功能**

根据输入的各进程资源总需求量、已分配资源、系统总资源，来判断某一进程接下来请求资源的操作是否会产生死锁，如果产生，就撤回操作；如果不产生，则给予资源并输出安全操作的进程序列。

* 1. **设计思路**

按照银行家制定的规则为进程分配资源，当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。当进程在执行中继续申请资源时，先测试该进程本次申请的资源数是否超过了该资源所剩余的总量。若超过则拒绝分配资源，若能满足则按当前的申请量分配资源，否则也要推迟分配。

一进程请求资源时，先判断是否会使系统产生死锁（所有进程能否安全执行），不产生，修改系统的各参数；产生，撤回刚才的资源。

* 1. **源代码**

1. #include <iostream>
2. **using** **namespace** std;
3. #define M 5                   //进程数量
4. #define N 3                   //资源种类
5. **int** Max[M][N] = { {5,5,9},{5,3,6},{3,0,8},{4,2,5},{4,2,4} };        //各进程对各种资源总的需求量
6. **int** Available[N] = { 2,3,3 };                                       //当前系统可用的各种资源量
7. **int** Allocation[M][N] = { {2,1,2},{4,0,2},{3,0,5},{2,0,4},{3,1,4} }; //各进程当前已分配到的各类资源
8. **int** Need[M + 1][N] = { {3,4,7},{1,3,4},{0,0,3},{2,2,1},{1,1,0} };   //各进程对资源的剩余需求量
9. **int** Request[N] = { 0,0,0 };                                         //当前某进程请求的各资源量
11. **void** show()   //显示系统现状
12. {
13. cout << "系统可用资源数为:";
14. **for** (**int** j = 0; j < N; j++) {
15. cout << Available[j] << " ";
16. }
17. cout << endl;
18. cout << "各进程的对资源的总需求" << endl;;
19. **for** (**int** i = 1; i <= M; i++) {
20. cout << "进程" << i << ":";
21. **for** (**int** j = 0; j < N; j++) {
22. cout << Max[i - 1][j] << " ";
23. }
24. cout << endl;
25. }
26. cout << "各进程已得到的资源" << endl;
27. **for** (**int** i = 1; i <= M; i++) {
28. cout << "进程" << i << ":";
29. **for** (**int** j = 0; j < N; j++) {
30. cout << Allocation[i - 1][j] << " ";
31. }
32. cout << endl;
33. }
34. cout << "各进程还需要的资源" << endl;
35. **for** (**int** i = 1; i <= M; i++) {
36. cout << "进程" << i << ":";
37. **for** (**int** j = 0; j < N; j++) {
38. cout << Need[i - 1][j] << " ";
39. }
40. cout << endl;
41. }
42. cout << endl;
43. }
45. **void** chang(**int** i) {    //当请求被允许时，修改系统各参数
46. **for** (**int** j = 0; j < N; j++) {
47. Available[j] = Available[j] - Request[j];
48. Allocation[i][j] = Allocation[i][j] + Request[j];
49. Need[i][j] = Need[i][j] - Request[j];
50. }
51. }
53. **void** rstor(**int** i) {    //当请求不被允许时，恢复系统各参数
54. cout << "该请求将使系统产生死锁，拒绝执行!" << endl;
55. **for** (**int** j = 0; j < N; j++) {
56. Available[j] = Available[j] + Request[j];
57. Allocation[i][j] = Allocation[i][j] - Request[j];
58. Need[i][j] = Need[i][j] + Request[j];
59. }
60. }
62. **int** chkerr() {      //当有请求时，判断是否会使系统产生死锁，若否，输出当前的安全序列
63. **int** Finish[M] = { 0 };
64. **int** Result[M] = { 0 };
65. **int** Work[N];
66. **int** sum = 0;
67. **for** (**int** i = 0; i < N; i++)
68. Work[i] = Available[i];
69. **while** (**true**) {
70. **int** flag1, flag2 = 0;
71. **for** (**int** i = 0; i < M; i++) {
72. flag1 = 1;
73. **if** (Finish[i] == 0) {
74. **for** (**int** j = 0; j < N; j++) {
75. **if** (Need[i][j] > Work[j]) flag1 = 0;
76. }
77. **if** (flag1) {
78. flag2 = 1;
79. **for** (**int** j = 0; j < N; j++)
80. Work[j] = Work[j] + Allocation[i][j];
81. Finish[i] = 1;
82. Result[sum] = i + 1;
83. sum++;
84. }
85. }
86. }
87. **if** (sum == M) {
88. cout << "当前系统的安全序列为:";
89. **for** (**int** i = 0; i < M; i++) {
90. cout << Result[i] << " ";
91. }
92. cout << endl;
93. cout << endl;
94. **return** 0;
95. }
96. **return** 1;
97. }
98. }
99. }
101. **int** run(**int** flag) {   //方便进行循环操作
102. **int** i = 0;
103. show();
104. **if** (flag == 1) {
105. cout << "输入需申请资源的进程号:";
106. cin >> i;
107. cout << "请输入进程" << i << "申请的资源数" << endl;
108. i--;
109. cout << "资源A、B、C:";
110. cin >> Request[0] >> Request[1] >> Request[2];
111. cout << endl;
112. chang(i);
113. **if** (chkerr()) {
114. rstor(i);
115. show();
116. }
117. **else**
118. show();
119. cout << "是否继续演示银行家算法，1是，0否:";
120. cin >> flag;
121. }
122. **return** flag;
123. }
124. **int** main() {
125. **int** a = 1;
126. a = run(a);
127. **while** (a == 1)
128. a = run(a);
129. **return** 0;
130. }
     1. **测试**

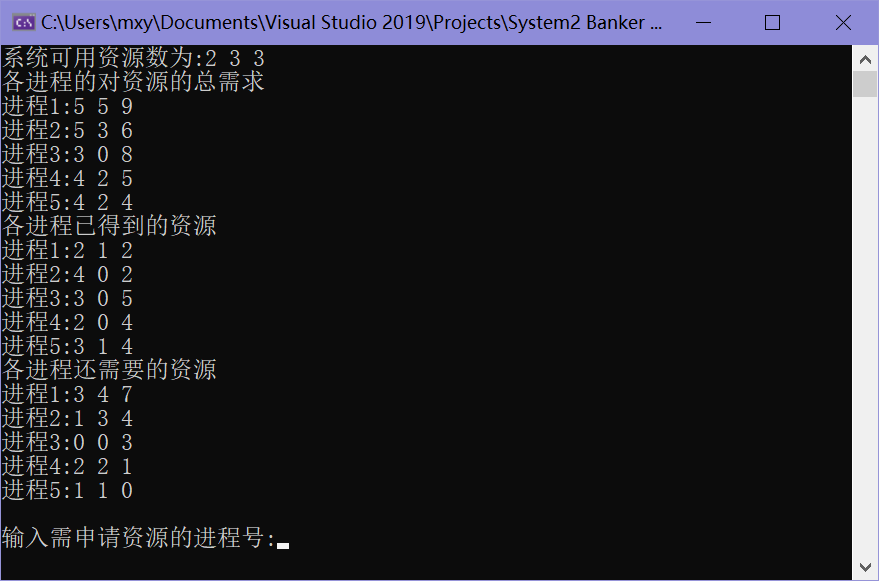
**测试数据：**

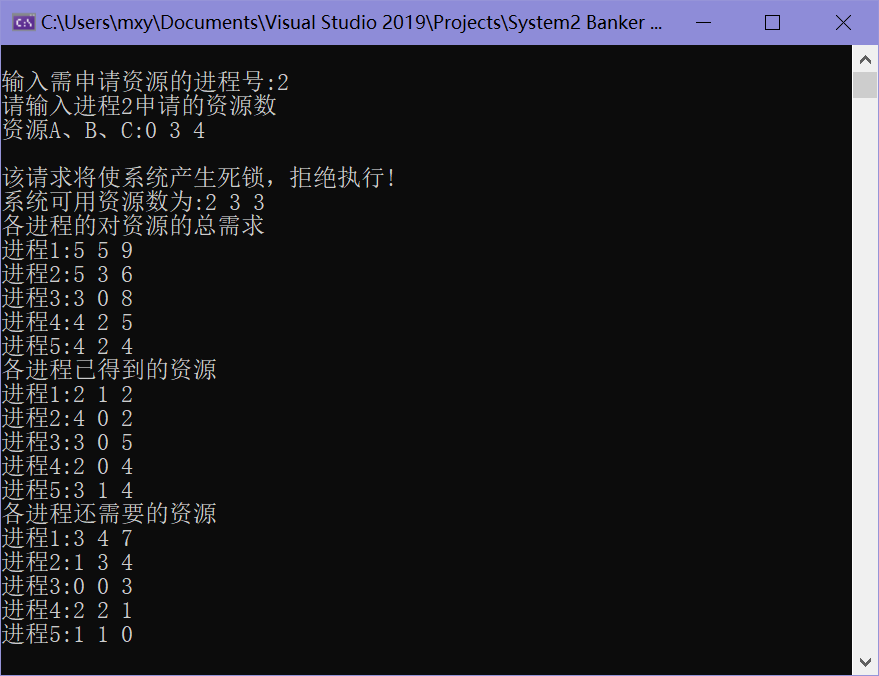
* m=3；
* 种类型的资源（A,B,C,）；
* 进程个数n=5；
* Available=(2,3,3);

|  |  |  |
| --- | --- | --- |
|  | 已分配资源数量 | 资源需求量 |
| A B C | A B C |
| P1 | 2 1 2 | 3 4 7 |
| P2 | 4 0 2 | 1 3 4 |
| P3 | 3 0 5 | 0 0 3 |
| P4 | 2 0 4 | 2 2 1 |
| P5 | 3 1 4 | 1 1 0 |

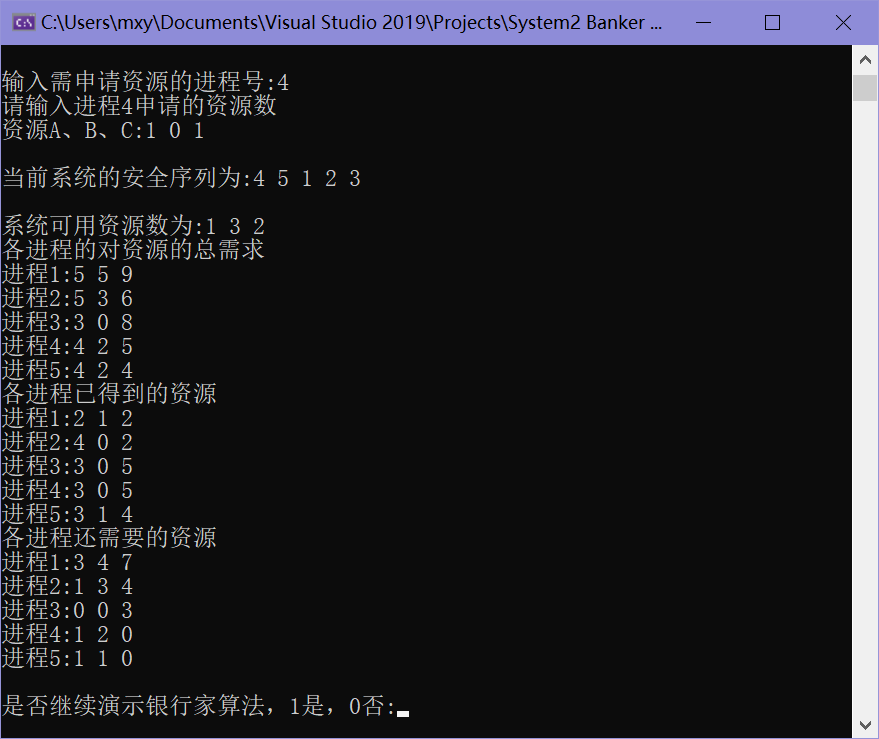
请求序列如下：

1. 进程P2请求资源（0，3，4）
2. 进程P4请求资源（1，0，1）
3. 进程P1请求资源（2，0，1）
4. 进程P3请求资源（0，0，2）

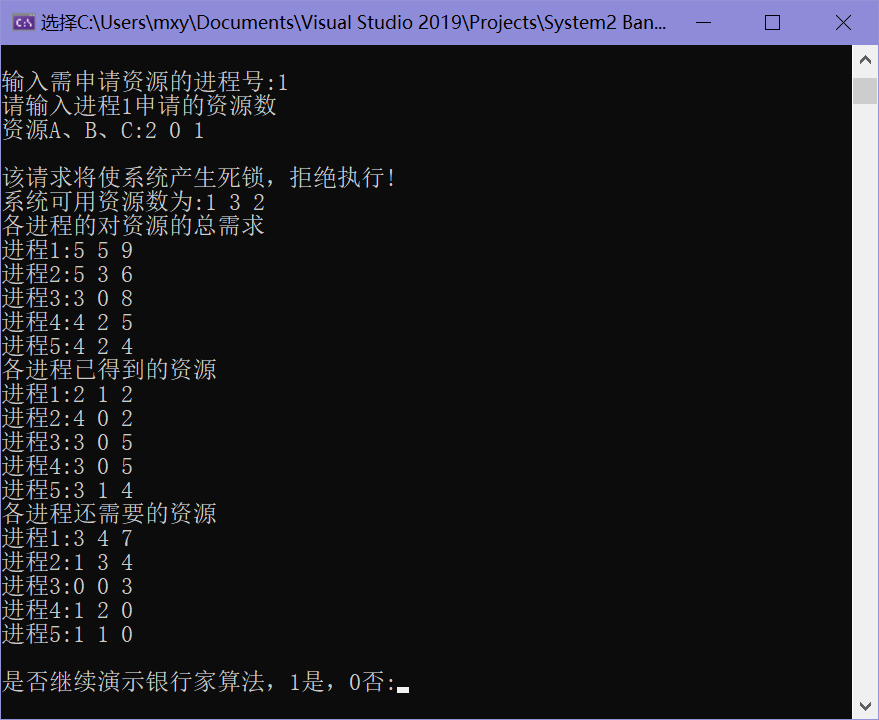


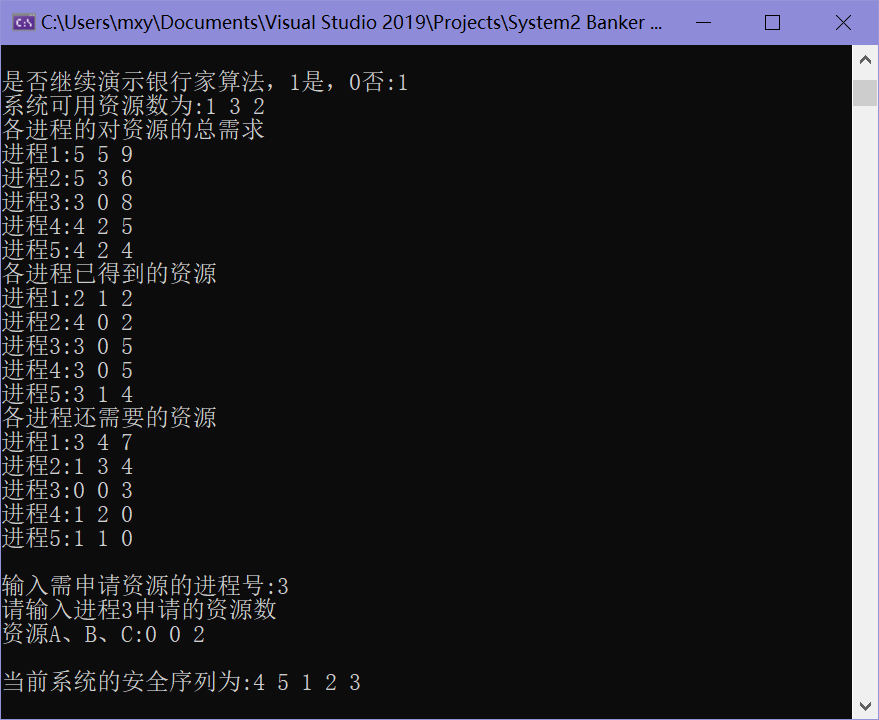


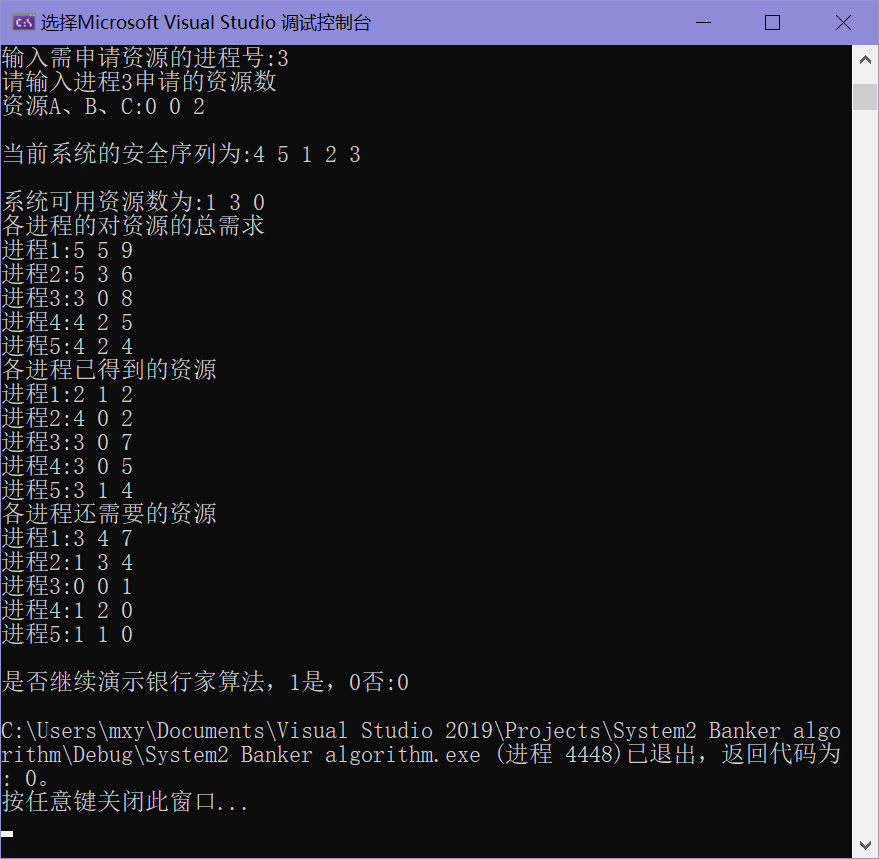












* 1. **实习心得**

通过这次实习，我对银行家算法和死锁都有了一定的理解。知道了在资源一定的条件下为了让多个进程都能使用资源完成任务，避免死锁的产生可以从一开始就对系统进行安全性检查来判断是否将资源分配给正在请求的进程。但是银行家算法会加大系统的开销。

并发和共享操作系统的基本特征，系统允许多个进程并发执行，并且共享系统的软、硬件资源。为了最大限度的利用计算机资源，操作系统应采用动态分配的策略，但是这样就容易因资源不足、分配不当而引起“死锁”。

**实验三 动态分区式存贮区管理**

* 1. **实习题目**

设计一个动态分区式存贮区管理程序，支持不同的放置策略。

* 1. **程序功能**
* 申请内存
* 释放内存
* 查看当前系统内存分配情况
* 循环执行上述操作
  1. **设计思路**

进程请求内存时，寻找内存空闲区，若空闲区富裕，则整体分配，否则，在释放内存的区域中需找空闲区，必要时应当切割空闲区。

进程释放内存时，判断进程是否有邻接的空闲区，若有则合并。

* 1. **源代码**

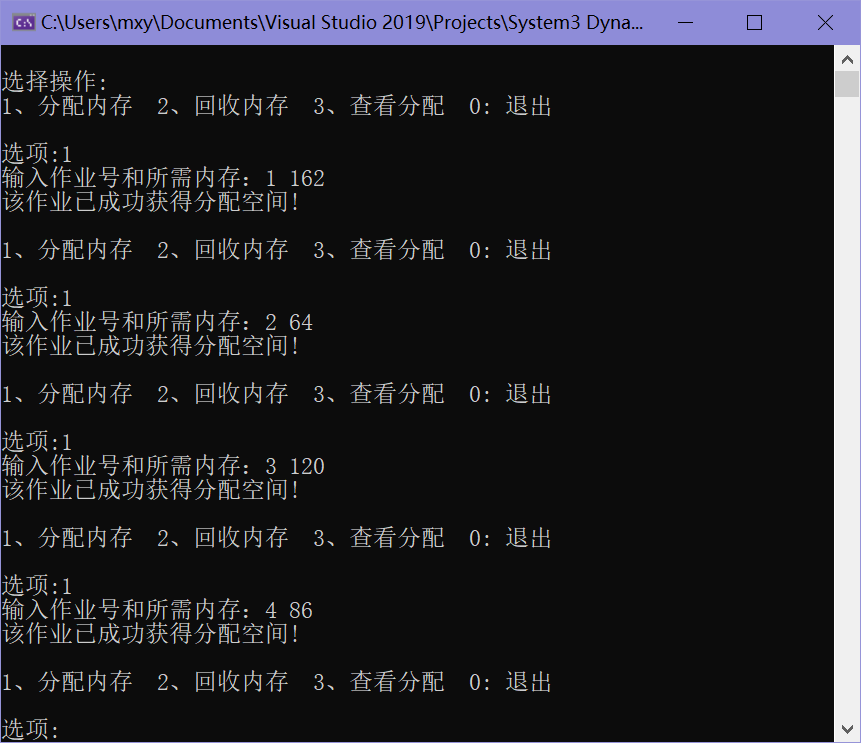
1. #include <iostream>
2. #include <iomanip>
3. **using** **namespace** std;
5. **int** maxsize = 512;
6. **int** minsize = 5;     //碎片最小值
7. **int** count1 = 0;
8. **int** count2 = 0;
9. #define  M  10     //系统允许的空闲区表最大为m
10. #define  N  10       //系统允许的最大作业数量为n
12. **struct** {             //已分配表
13. **float** address;   //已分配区起始地址
14. **float** len;       //已分配区长度
15. **int** flag;        //已分配区表登记栏标志，"0"表示空栏目
16. }used[N];
18. **struct** {             //空闲区表
19. **float** address;   //空闲区起始地址
20. **float** len;       //空闲区长度
21. **int** flag;        //空闲区表登记栏标志，用"0"表示空栏目，用"1"表示未分配
22. }frees[M];

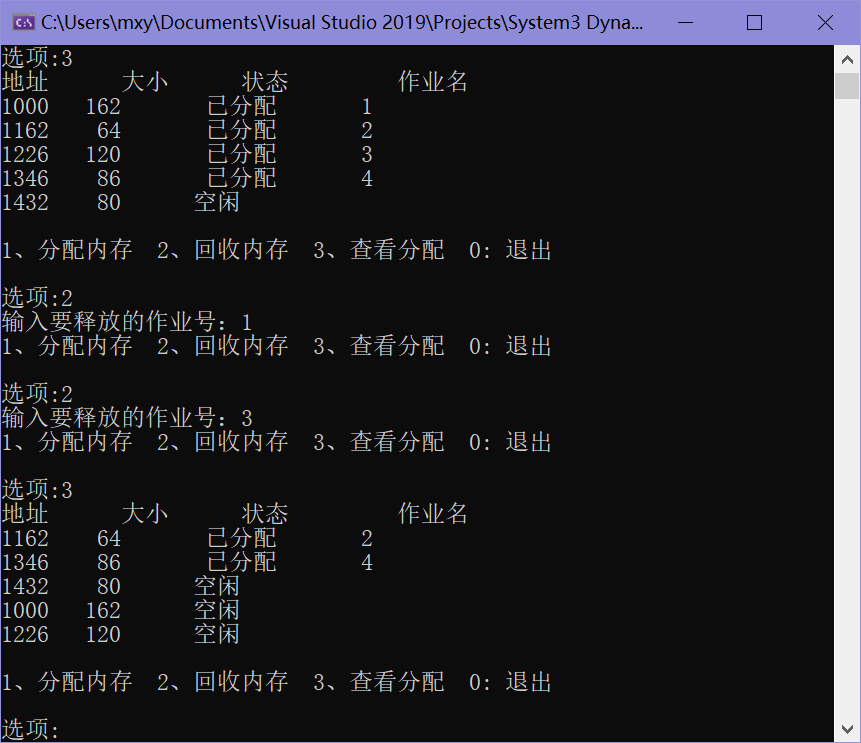
25. **void** init(**void**) {    //初始化两个表
26. **int** a;
27. **for** (a = 0; a <= N - 1; a++)
28. used[a].flag = 0;
29. frees[0].address = 1000;
30. frees[0].len = maxsize;
31. frees[0].flag = 1;
32. }
34. **int** distribute(**int** name, **float** n\_len) {    //动态分区
35. **int** i, k = -1;
36. **float**  ads, len;
37. **int** count = 0;
38. i = 0;
39. **while** (i <= M - 1) {
40. **if** (frees[i].flag == 1 && n\_len <= frees[i].len) {
41. count++;
42. **if** (count == 1 || frees[i].len < frees[k].len)
43. k = i;
44. }
45. i = i + 1;
46. }
47. **if** (k != -1) {
48. **if** ((frees[k].len - n\_len) <= minsize)
49. {
50. frees[k].flag = 0;
51. ads = frees[k].address;
52. len = frees[k].len;
53. }
54. **else** {
55. ads = frees[k].address;
56. len = n\_len;
57. frees[k].address += n\_len;
58. frees[k].len -= n\_len;
59. }
60. i = 0;
61. **while** (used[i].flag != 0) {
62. i = i + 1;
63. }
64. **if** (i <= N - 1) {
65. used[i].address = ads;
66. used[i].len = len;
67. used[i].flag = name;
68. count1++;
69. }
70. **else** {
71. **if** (frees[k].flag == 0) {
72. frees[k].flag = 1;
73. frees[k].address = ads;
74. frees[k].len = len;
75. }
76. **else** {
77. frees[k].address = ads;
78. frees[k].len += len;
79. }
80. cout << "内存分配区已满，分配失败！\n";
81. **return** 0;
82. }
83. }
84. **else** {
85. cout << "无法为该作业找到合适分区！\n";
86. **return** 0;
87. }
88. **return** name;
89. }
91. **int** release(**int** name)   //释放函数
92. {
93. **int** y = 0;
94. **float** ads, len;
95. **int** i, j, k;
96. **int** x;
97. **while** (y <= N - 1 && used[y].flag != name) {
98. y = y + 1;
99. }
100. **if** (y <= N - 1) {
101. ads = used[y].address;
102. len = used[y].len;
103. used[y].flag = 0;
104. count2++;
105. }
106. **else** {
107. cout << "该作业不存在！\n";
108. **return** 0;
109. }
110. j = k = -1;
111. i = 0;
112. **while** (!(i >= M || (k != -1 && j != -1))) {
113. **if** (frees[i].flag == 1) {
114. **if** ((frees[i].address + frees[i].len) == ads)
115. k = i;
116. **if** ((ads + len) == frees[i].address)
117. j = i;
118. }
119. i = i + 1;
120. }
122. **if** (k != -1)
123. {
124. **if** (j != -1) {
125. frees[k].len += frees[j].len + len;
126. frees[j].flag = 0;
127. }
128. **else**
129. frees[k].len += len;
130. }
131. **else** **if** (j != -1) {
132. frees[j].len += len;
133. frees[j].address = ads;
134. }
135. **else** {
136. x = 0;
137. **while** (frees[x].flag != 0)
138. x = x + 1;
139. **if** (x <= M - 1) {
140. frees[x].address = ads;
141. frees[x].len = len;
142. frees[x].flag = 1;
143. }
144. **else** {
145. used[y].flag = name;
146. cout << "空闲区已满，回收失败！\n";
147. **return** 0;
148. }
149. }
150. **return** name;
151. }
153. **void** show() {     //显示内存分配情况
154. cout << "地址" << setw(10) << "大小" << setw(10) << "状态" << setw(15) << "作业名" << endl;
155. **for** (**int** j = 0; j < count1; j++)
156. **if** (used[j].flag != 0)
157. cout << used[j].address << setw(6) << used[j].len << setw(13) << "已分配" << setw(8) << used[j].flag << endl;
158. **for** (**int** i = 0; i <= count2; i++)
159. **if** (frees[i].flag != 0)
160. cout << frees[i].address << setw(6) << frees[i].len << setw(10) << "空闲" << endl;
161. cout << endl;
162. }
163. **int** main() {
164. **int** a;
165. **int** name;
166. **float** memory;
167. **bool** flag = **false**;
168. cout << endl;
169. cout << "选择操作:\n";
170. init();
171. **while** (!flag) {
172. cout << "1、分配内存  2、回收内存  3、查看分配  0、退出 \n";
173. cout << endl;;
174. cout << "选项:";
175. cin >> a;
176. **if** (a == 0)
177. flag = **true**;
178. **else** **if** (a == 1) {
179. cout << "输入作业号和所需内存：";
180. cin >> name >> memory;
181. **if** (name != 0 && memory != 0) {
182. distribute(name, memory);
183. cout << "该作业已成功获得分配空间!" << endl;
184. cout << endl;
185. }
186. **else** **if** (name == 0)
187. cout << "作业号不能为零!\n请重新选择操作：\n";
188. **else** **if** (memory == 0)
189. cout << "内存分配数不能为零!\n请重新选择操作：\n";
190. }
191. **else** **if** (a == 2) {
192. **int** ID;
193. cout << "输入要释放的作业号：";
194. cin >> ID;
195. **if** (ID != 0)
196. release(ID);
197. }
198. **else** {
199. show();
200. }
201. }
202. **return** 0;
203. }

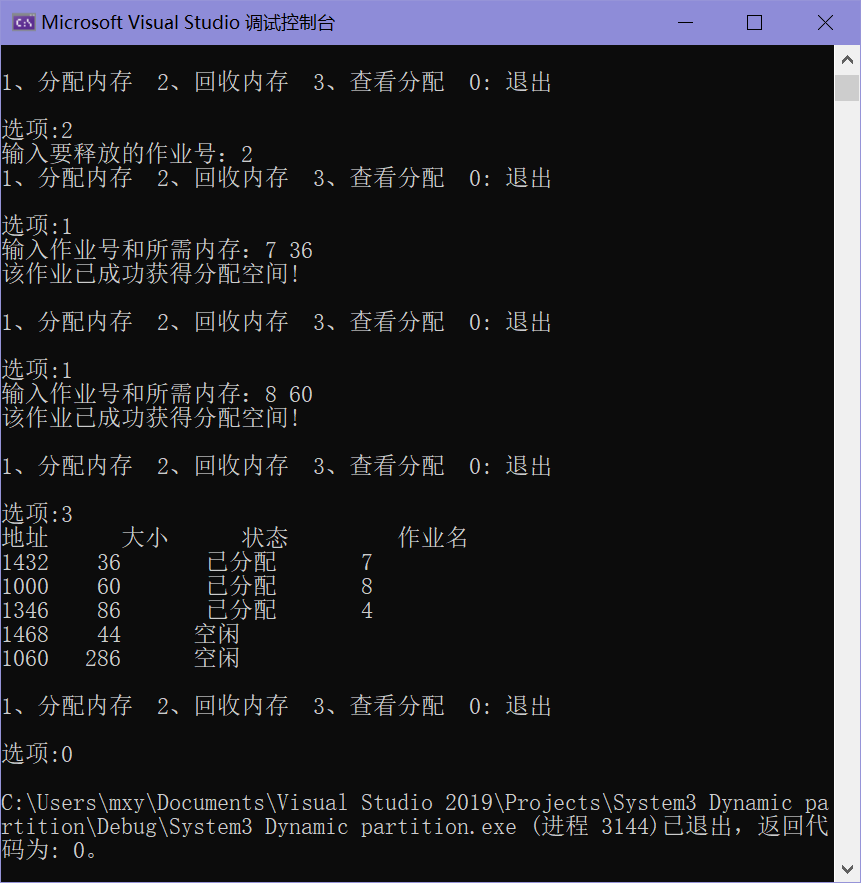
#include <iostream>

1. #include <iomanip>
2. **using** **namespace** std;
4. **int** maxsize = 512;
5. **int** minsize = 5;     //碎片最小值
6. **int** count1 = 0;
7. **int** count2 = 0;
8. #define  M  10       //系统允许的空闲区表最大为m
9. #define  N  10       //系统允许的最大作业数量为n
11. **struct** {             //已分配表
12. **float** address;   //已分配区起始地址
13. **float** len;       //已分配区长度
14. **int** flag;        //已分配区表登记栏标志，"0"表示空栏目
15. }used[N];
17. **struct** {             //空闲区表
18. **float** address;   //空闲区起始地址
19. **float** len;       //空闲区长度
20. **int** flag;        //空闲区表登记栏标志，用"0"表示空栏目，用"1"表示未分配
21. }frees[M];

24. **void** init(**void**) {    //初始化两个表
25. **int** a;
26. **for** (a = 0; a <= N - 1; a++)
27. used[a].flag = 0;
28. frees[0].address = 1000;
29. frees[0].len = maxsize;
30. frees[0].flag = 1;
31. }
33. **int** distribute(**int** name, **float** n\_len) {    //动态分区
34. **int** i, k = -1;
35. **float**  ads, len;
36. **int** count = 0;
37. i = 0;
38. **while** (i <= M - 1) {
39. **if** (frees[i].flag == 1 && n\_len <= frees[i].len) {
40. count++;
41. **if** (count == 1 || frees[i].len < frees[k].len)
42. k = i;
43. }
44. i = i + 1;
45. }
46. **if** (k != -1) {
47. **if** ((frees[k].len - n\_len) <= minsize)
48. {
49. frees[k].flag = 0;
50. ads = frees[k].address;
51. len = frees[k].len;
52. }
53. **else** {
54. ads = frees[k].address;
55. len = n\_len;
56. frees[k].address += n\_len;
57. frees[k].len -= n\_len;
58. }
59. i = 0;
60. **while** (used[i].flag != 0) {
61. i = i + 1;
62. }
63. **if** (i <= N - 1) {
64. used[i].address = ads;
65. used[i].len = len;
66. used[i].flag = name;
67. count1++;
68. }
69. **else** {
70. **if** (frees[k].flag == 0) {
71. frees[k].flag = 1;
72. frees[k].address = ads;
73. frees[k].len = len;
74. }
75. **else** {
76. frees[k].address = ads;
77. frees[k].len += len;
78. }
79. cout << "内存分配区已满，分配失败！\n";
80. **return** 0;
81. }
82. }
83. **else** {
84. cout << "无法为该作业找到合适分区！\n";
85. **return** 0;
86. }
87. **return** name;
88. }
90. **int** release(**int** name)   //释放函数
91. {
92. **int** y = 0;
93. **float** ads, len;
94. **int** i, j, k;
95. **int** x;
96. **while** (y <= N - 1 && used[y].flag != name) {
97. y = y + 1;
98. }
99. **if** (y <= N - 1) {
100. ads = used[y].address;
101. len = used[y].len;
102. used[y].flag = 0;
103. count2++;
104. }
105. **else** {
106. cout << "该作业不存在！\n";
107. **return** 0;
108. }
109. j = k = -1;
110. i = 0;
111. **while** (!(i >= M || (k != -1 && j != -1))) {
112. **if** (frees[i].flag == 1) {
113. **if** ((frees[i].address + frees[i].len) == ads)
114. k = i;
115. **if** ((ads + len) == frees[i].address)
116. j = i;
117. }
118. i = i + 1;
119. }
121. **if** (k != -1)
122. {
123. **if** (j != -1) {
124. frees[k].len += frees[j].len + len;
125. frees[j].flag = 0;
126. }
127. **else**
128. frees[k].len += len;
129. }
130. **else** **if** (j != -1) {
131. frees[j].len += len;
132. frees[j].address = ads;
133. }
134. **else** {
135. x = 0;
136. **while** (frees[x].flag != 0)
137. x = x + 1;
138. **if** (x <= M - 1) {
139. frees[x].address = ads;
140. frees[x].len = len;
141. frees[x].flag = 1;
142. }
143. **else** {
144. used[y].flag = name;
145. cout << "空闲区已满，回收失败！\n";
146. **return** 0;
147. }
148. }
149. **return** name;
150. }
152. **void** show() {     //显示内存分配情况
153. cout << "地址" << setw(10) << "大小" << setw(10) << "状态" << setw(15) << "作业名" << endl;
154. **for** (**int** j = 0; j < count1; j++)
155. **if** (used[j].flag != 0)
156. cout << used[j].address << setw(6) << used[j].len << setw(13) << "已分配" << setw(8) << used[j].flag << endl;
157. **for** (**int** i = 0; i <= count2; i++)
158. **if** (frees[i].flag != 0)
159. cout << frees[i].address << setw(6) << frees[i].len << setw(10) << "空闲" << endl;
160. cout << endl;
161. }
162. **int** main() {
163. **int** a;
164. **int** name;
165. **float** memory;
166. **bool** flag = **false**;
167. cout << endl;
168. cout << "选择操作:\n";
169. init();
170. **while** (!flag) {
171. cout << "1、分配内存  2、回收内存  3、查看分配  0、退出 \n";
172. cout << endl;;
173. cout << "选项:";
174. cin >> a;
175. **if** (a == 0)
176. flag = **true**;
177. **else** **if** (a == 1) {
178. cout << "输入作业号和所需内存：";
179. cin >> name >> memory;
180. **if** (name != 0 && memory != 0) {
181. distribute(name, memory);
182. cout << "该作业已成功获得分配空间!" << endl;
183. cout << endl;
184. }
185. **else** **if** (name == 0)
186. cout << "作业号不能为零!\n请重新选择操作：\n";
187. **else** **if** (memory == 0)
188. cout << "内存分配数不能为零!\n请重新选择操作：\n";
189. }
190. **else** **if** (a == 2) {
191. **int** ID;
192. cout << "输入要释放的作业号：";
193. cin >> ID;
194. **if** (ID != 0)
195. release(ID);
196. }
197. **else** {
198. show();
199. }
200. }
201. **return** 0;
202. }
     1. **测试**







* 1. **实习心得**

存储数据是操作系统很重要的功能之一，而要保证数据存储的高效以及提高磁盘资源的利用率，对存储区的管理就显得十分的重要。文件总是数量巨大并且各个文件大小不一，所占用的硬盘空间也各不相同，不加管理的任其存放最终就会导致磁盘空间的碎片化严重，极其浪费硬件资源。在对存储区管理的学习中，了解到了各种管理算法，也让我对操作系统的磁盘管理有了更深刻的理解。一个好的分区管理算法，不仅可以提高磁盘效率，并且也可以明显的提升磁盘资源的利用率。

可见，写代码并不是最重要的，重要的是理解算法，学习算法，培养一种计算机的思维，只有充分理解了算法，才能写出正确、高效的程序。